Moving up the Product Security Maturity Model

Authors: Joshua Cajetan Rebelo and Patrick McEnany

joshua.cajetan.rebelo@intel.com patrick.s.mcenany@intel.com

Abstract

The world has become a global village, and it is being ruled and prominently controlled by technology and electronics in particular. This results to consistent increase in the availability of personal, corporate, and financial information in cyberspace. This creates enormous opportunities for cyber attackers to access the data and misuse it through hacking tools and tutorials.

One such recent example is the intellectual property theft in the Xbox One gaming console and Xbox Live. The hacking of Target and Home Depot networks lead to the leakage of sensitive data such as email-ids and credit card details. Another example is the data breach of 4.2 million individuals in the US Government Office of Personnel Management (OPM). These incidents clearly emphasize the necessity to deliver a comprehensive secure product. For organizations, the goal must be to adopt a better strategy and protect the data and resources in a more proactive manner.

Organizations span the spectrum when it comes to the maturity around creating secure products. Some organizations have a well-defined process that ensures the delivery of highly secure products whereas some organizations want to improve the security maturity model but lack management support. Some organizations are not even aware of product security and they are not sure from where to start.

This paper defines a multi-layered security approach that can be applied to any platform, product, and programming language. The multi-layered security helps the product teams that having little knowledge of product security to uncover the low hanging security defects. As the team gains expertise they become Evangelist and Champions of secure software development.

Since the threat types and attack vectors are evolving at a rapid pace, creating a security maturity model for the product that can provide up-to-date protection and realign its capability to handle the latest security challenges are vital.

Biography

Joshua Cajetan Rebelo is a Senior Technical Lead and a Product Security Champion at Intel Security Group with 11 years of experience in the security domain. He ensures that the products are under continuous development to incorporate the highest Security Software Development Lifecycle (SDL) standards and adhere to the Product Security Maturity Model (PSMM) for pro-active in findings vulnerabilities. He has vast experience in conducting security code reviews and audits, evaluating software architecture for potential risk, and recommending design changes to address security concerns. He also brings innovation with new ideas and has three patents issued to his credit.

Patrick McEnany is a Senior Manager and a Lead Product Security Champion for the Intel Security Group (ISecG) with 18 years of experience in the software product security domain. He is an extended core team member of the ISecG Product Security Group and directly manages the product security programs across 17 core Intel Security Enterprise Products. Many of his primary responsibilities include adoption and enforcement of the Product Security Maturity Model (PSMM) and Product Security Incident Response Team (PSIRT) program. He is continuously working with products teams to mentor volunteers designated as Product Security Champions at the product team level.

Intended Audience

The target audience for this paper includes program and project managers, developers, quality assurance engineers, and IT professional who are familiar with the software security literature. This paper is also meant for people who are passionate about security and want to integrate security into their standard software development lifecycle (SDL) processes and to enhance the product security maturity model to stay secured.

1. Introduction

Many organizations do not plan product security until the end of the overall development process. Instead of planning and executing it for all phases of the release, they run the external security audits at the end. Following the traditional Waterfall method at the end of a development release will always uncover security issues such as incorrect input validation, SQL injection, XSS attacks, weak SSL design, or vulnerable third-party libraries with known vulnerabilities. Finding the security issues at the end of the cycle always hits the release schedule. If the security issues are left unresolved, it increases the risk for the product, affects the brand recognition in the industry, and financial loss for organization, and the customer loyalty. Critical security flaws often require architectural design changes that in turn increase your development costs by refactoring the components involved.

Some organizations think that product security can be addressed through a check-list. This will actually not always work out because attacks are evolving and becoming more sophisticated as time progresses.

Some organizations might follow a reactive approach by responding to vulnerabilities only when they are reported. This is the most dangerous model, because the “Bad Guys” will conceal any successful attacks against your products as long as possible to exploit the weakness for years before the issue is fully resolved.

This paper will help product teams to achieve the following product security goals using the maturity model.

- Allocate resources, set aside budget and create unique titles / roles within the team.
- Identify the security engineers who play a vital role in addressing the security issues of the product.
- Identify the job responsibilities of the security engineer.
- Help product teams to fit themselves into the maturity model and to get started with product security.
- Product teams who are already addressing security in bits and pieces will follow the guidelines and implement the security maturity model explained below to identify risks/security flaws earlier in the SDL in an organized manner.
- Addressing product security throughout the product development lifecycle right from the concept phase to architecture, planning, design, and implementation phase.
- Achieve higher benchmarks of product security.

2. Understanding the Security Posture

To understand the security posture of the product, the Security Engineer and the Product Architect should come up with the security requirements.

2.1 What is Security Requirement?

Security Requirements describe functional and non-functional requirements that need to be satisfied to achieve the security attributes of a system.

When the security requirements are clearly defined at the onset of the project, it allows the development teams to trade-offs cost of baking security into a product. This will define the scope to build a strong defense in depth for the product.

At the onset of the project, the Definition of Done (DoD) criteria should make the goals of the security requirement **S.M.A.R.T**

Specific	Target a specific area for improvement
Measurable	Quantifiable or at least suggest an indicator of progress
Attainable	Requirements should be achievable by developing the required abilities and skills
Realistic	Requirements should be realistically achievable
Timely	Requirements should be grounded within a time frame

DoD is a minimum set of tasks that need to be defined and executed to mark the status of security requirement as completed. Depending on the DoD criteria, the security requirement can be classified as one time activity or a continuous activity.

One Time Activity: A requirement that is implemented once can be verified and marked as completed according to the DoD. Example: A strong password.

Continuous Activity: A requirement that has a DoD but is a continuous activity that needs to be executed for every build drop or at every release milestones. This type of requirement is important because vulnerabilities are introduced as and when new codes are checked in. For example, running vulnerability scans and performing code static analysis on every build that is delivered to QA for testing.

2.1.1 Examples of Security Requirements

The following are some of the most applicable security requirements that can be classified as one time or continuous activity. The details of these security requirements are explained in the maturity model below. Security requirements can vary based on the user and product requirements.

Sl. No	Security Requirement	One Time	Continuous
1	Resource/Headcount Identification & Budgeting	✓	
2	Mandatory Trainings/Certification	✓	✓
3	Product Hardening		✓
4	Secure Code Standards		✓
5	Code Review		✓
6	Vulnerability Assessment	✓	✓
7	Static Analysis		✓
8	Open Source / Third Party Library Review	✓	✓
9	Compliance	✓	✓
10	Secure Storage of sensitive data	✓	✓
11	Common Vulnerability Scoring System (CVSS)		✓
12	External reported security flaws		✓
13	Security Bulletins and KnowledgeBase articles	✓	
14	Secure Functional Requirements		✓
15	Cryptographic Support Requirement		✓
16	Least privilege implementation	✓	✓
17	Threat Modeling	✓	✓
18	Secure Architecture Review	✓	✓
19	Product Fuzzing		✓
20	Reverse-Engineering		✓
21	Writing Exploits		✓
22	Security Test Plan	✓	✓

In the above table, some of the requirements are tagged as both one time and continuous activities. For example, the Security Test Plan. This is because as the product matures and the product team moves up the security maturity model, the requirements need to be re-visited, reviewed and the DoD criteria needs to be updated.

The finalized security requirements need to be reviewed with the product team by asking the following questions:

- Are we doing the right things to address the product security?
- Have the right set of tools and people been identified to accomplish these security requirements according to the DoD?
- Are these security goals S.M.A.R.T?
- Will the DoD of the requirement help to move up the product security maturity model by setting higher benchmarks for security?
- Is the product team confident that the product will become more secured with the identified security requirements?
- After executing all security requirements, is the product team confident that there will be a drastic reduction of security issues in the product?
- Is the approach reactive or proactive?
- Will the security requirements help to identify the risks found earlier in the SDL?
- Will the security requirements help the product to be one step ahead of the “Bad Guys”?
- Do the security requirements protect the product against the latest attacks?
- You can find good security requirement checklists in the web. For example, CERN security checklist for developers.

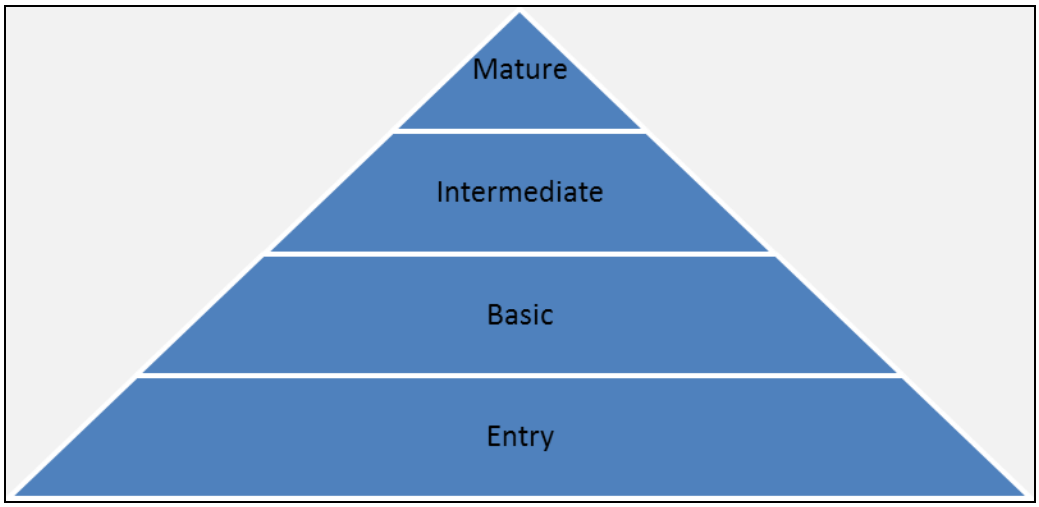
2.1.2 Responsibilities of the Security Engineer

The Security Engineer is the one who is a single point of contact for all security-related issues. When it comes to product security, the Security Engineer is the eyes and ears for the team.

- Work with the Product Architect to identify security requirements.
- Prioritize, plan, and execute all security requirements.
- Convert the identified security issues or flaws to user stories and update the project backlog.
- Prioritize all security user stories in project backlog and work closely with the product team to complete them.
- Conduct periodic security reviews.
- Assess and confirm the reported security issues and analyze the risk involved by doing the CVSS scoring.
- Create Security Bulletins for all reported security issues that have a CVSS score 4.0 and later.
- Create KnowledgeBase articles for any security issue having CVSS score lesser than 4.0.
- Review and update the Threat model for missing attack vectors.
- Be the eyes, ears, and advocate of security by becoming an influencing Security Evangelist within the team and the business unit.
- Participate in all security-related meetings and discussions.
- Address all security-related escalations.

3. Product Security Maturity Model

The product security maturity model level can be classified into four groups.



The approach followed in this paper is to start with the requirements that are easy to execute and that helps to get started with security. As you gain experience and become an expert in security it gets into your DNA. Then you move up the product security maturity model.

This paper focuses on achieving goals at the lower levels, then move up to the higher levels of maturity model to achieve tougher goals. This progressive approach will help the team as a whole to mature in product security. If the team works at all levels simultaneously, it becomes difficult to rate and position the team in terms of the maturity level of their product security.

Complex requirements can be broken down into two parts namely basic and complex parts, and it can be addressed across the maturity levels. The basic part can take care of implementing the basic security at the lower levels. As expertise increases the complex part of the requirement can be implemented at the higher levels. For example, simple fuzzing and basic SSL support can be implemented at lower levels and the complex part is implemented at higher levels.

Depending on the nature of the requirement, it needs to be executed by:

- The Security Engineer alone (For example: doing a third-party library audit, performing vulnerability assessment, etc.)
- An individual engineer (For example: secure coding practices by each development engineer)
- All members of the product team (For example: code review for security)

To assess the progress of each requirement, the Security Engineer must conduct regular review meetings with the product team throughout the development lifecycle. The meeting objective should be to track the progress made on each requirement by rating them as:

- None
- Initial
- Basic
- Acceptable
- Mature

For the detailed parameters to measure these levels, see, **Appendix**.

Rating security requirement with these levels will help to achieve:

- Comfort level of the product team in understanding and executing the requirements.
- Overall progress made on each requirement.
- Positioning the product into the product security maturity model.
- Create room for improvement and optimization of the requirements.

3.1 Security Levels

3.1.1 Entry Level

At the entry level, the following tasks need to be executed to lay a strong foundation for the product security.

- Separate resources are allocated to address product security. For example
  - Budget approval for Security Engineer headcount
  - Budget allocated for security training for creating security awareness
  - Budget allocated to procure security tools

- The Security Engineer is identified for each product team. The mid and junior security engineers who will assist the Security Engineer can also be identified.

- Mandatory Security Trainings / Certificate: Arrange for an in-house or online training on security. Depending on the job responsibilities, mandatory training / certification needs to be assigned to the team including the Security Engineer to improve their skills and knowledge on security.

  The following are some of the trainings and certifications:

  - OWASP Top 10 vulnerabilities
  - WASC Threat Classification
  - Secure coding practices
  - Secure coding in Java / C / C++ and other languages
  - Practice sessions on WebGoat tool
  - Ethical Hacking (CEH)
  - Metasploit Certification

- The Security Engineer conducts a brainstorming session with the product team to identify all possible attack vectors. Some of the points to focus in this meeting are:
  - Enumerate all possible entry points to the system.
  - Enumerate all possible threats for the system and its intended deployment.
  - Enumerate all possible threat agents who would show interest in the system.
  - Enumerate the objective/goals of each of these threat agents.
  - Enumerate the typical attack methods of the threat agents.
  - Intersect attack methods against the entry points to get the set of attack surfaces.
  - List all existing security controls (mitigations) for each of the attack surfaces.
  - Filter the threat agents who have no attack surfaces exposed to their typical methods.
  - Filter all attack surfaces for which there is sufficient existing protection.

  This exercise will help the team to focus on security, and the Security Engineer can use this data while creating security requirements. This exercise needs to be done at all levels of the maturity model.

- The Security Engineer, the Product Architect, and the product team altogether need to harvest the security requirements for attack surfaces for which there is no sufficient mitigation.
- Segregate all harvested security requirements and integrate them into the product security maturity model which is explained in detail below.

3.1.2 Basic Level

The foundation is set after making it through the entry level. The next level in the maturity model is the basic level. At this level, the product team will have to work on the following set of security requirements which will help to uncover most of the low hanging security defects with little knowledge of security. At this level it's all about learning security and getting hands-on experience.

3.1.2.1 Product Hardening

Product hardening is a process of securing a system by reducing its surface of vulnerability by deploying and operating products in a secure manner. Reducing available attack surface typically involves the following:

- Run processes and services with the least privileges and not with the system or the root privileges.
- Protect the backup files with password.
- Authenticate all communications and data transactions between the product modules.
- Limit the data access of all tenant users to their own tenant scope in a Multi-Tenant mode.
- Review the default configuration and make sure that only the whitelisted ports and services are allowed to run.
- Uninstall the unwanted applications from the base operating system which is used for product installation.
- Review the third-party modules configuration and disable the defaults.

3.1.2.2 Secure Coding Standards

This requirement suggests following secure coding practices and coding guidelines. The mandatory trainings completed at the Entry Level will help the developers when implementing key takeaways from the security training. These best practices will take care of most of the input validation and other vulnerabilities at the development phase.

3.1.2.3 Code Review

All code needs to be reviewed for security. Teams can start doing peer code review then move on to a centralized tool which will allow multiple team members to participate in security code reviews. This in turn will help in early security defect detection.

3.1.2.4 Vulnerability Assessment

Vulnerability scanners need to be evaluated and procured. These scanners will help to uncover the basic security defects. Running these scanners does not need any expertise but clearing off the noise and false positives from the full report needs some basic understanding of security.

Burp Suite is a good web vulnerability scanner to identify the top 10 OWASP vulnerabilities. Some of the other top web vulnerability scanners are Acunetix, Rapid 7, Retina, Cenzic, McAfee Vulnerability scanner and OWASP ZAP proxy.

3.1.2.5 Static Analysis

There are many tools that you can use for static analysis. The tool selection depends on the coding language used in the product.

- Coverity is a good static analysis tool which can be integrated into the build system to run static analysis on every build. There are two components of coverity:
  - Server component – Runs analysis against every build delivered to QA.
  - Client component – Used by development teams to check if the code is written securely before committing the code to the configuration management system.
- FindBugs is another static analysis tool which can be used to identify security issues in Java.
- Checkmarx is another code analysis tool which supports multiple coding languages.

3.1.2.6 Open Source / Third-Party Library Review

Recent vulnerabilities such as Heartbleed, Freak Attack, and Logjam have increased the importance of conducting third-party library review. All third-party libraries that are used by the product must go through the security review process of known vulnerabilities. This review process calls for a proactive exercise of upgrading/updating all legacy libraries with new and safer versions of the libraries.

For example, upgrading the OpenSSL, Apache, and tomcat libraries which are of interest to the “Bad Guys”.

OWASP Dependency Checker is a good tool to find known vulnerabilities in third-party libraries.

3.1.2.7 Compliance

EU/PII Data protection, FFIEC, HIPAA, ISO 2700x, PCI, NCUA, FIPS compliance will help to embed security into the product from compliance point of view. But the maturity model should never be compliance driven. It should be one of the important requirements which need to be achieved in the product security model.

3.1.2.8 Secure Storage of Sensitive Data

All sensitive data collected or generated by the product module must be stored securely. Following are some of the pointers to secure the sensitive data.

- No hardcoding of credentials.
- Credentials stored in DB or system files should be encrypted.
- Data logged into log files should not contain sensitive information in clear text, it should be masked.
- Passwords or passphrases used to protect sensitive information should be non-static and unpredictable. It should be randomly generated and recycled periodically.

3.1.2.9 CVSS Scoring

CVSS scoring needs to be validated for all security flaws reported on the product using the CVSS calculator. CVSS scoring also needs to be done for security flaws reported from the field to reassess the risk because this score can either be inflated or kept low by the researchers or hackers. If a vulnerability is reported on any third-party library that is used by the product, the reported CVSS score needs to be re-validated. The risk introduced by these vulnerable libraries can either go up or come down depending on the implementation, impact, exposure, and distribution.

The following guidelines can be followed to address the scope of fixing the security flaws:

Risk Type	Score	Required Action
Critical	8.5 > 10.0	Must be fixed immediately.
High	7.0 > 8.4	Must be fixed within a week.
Medium	4.0 >6.9	Must be fixed within a month.
Low	0.0 >3.9	Must be fixed in the next major version.

3.1.2.10 External Reported Security Flaws

The Security Engineer should respond to security flaws reported from external sources by validating the CVSS score of the reported vulnerabilities as described in the CVSS scoring section and perform risk reassessment. This assessment will either increase or decrease the risk based on the following.

- **Not Vulnerable** — Risk or Impact is low. [Risk Score - None]
- **Vulnerable** — High Risk due to usage of vulnerable function / code. [Risk Score - Critical, High]

- **Vulnerable, but Not Exploitable** — Medium Risk since no usage of vulnerable code/function but flagged by vulnerability scanners due to version match. [Risk Score – Medium]
- **Vulnerable, but Low Risk** — Low Risk due to less impact, exposure and distribution. [Risk Score - Low]

Risk reassessment can create an opportunity to identify new entry points and generate new security requirement or update the existing requirements.

In some cases, responding to complex security flaws requires some level of expertise and understanding of the internal implementation of the product.

3.1.2.11 Privacy Impact

In some cases, products deployed in customer environment collect customer-related information and send it back to its manufacturer. This personally identifiable information (PII) needs to be safeguarded from the time it leaves the customers environment. Once it reaches the backend servers in the manufacturer network it needs to have certain safeguards mechanism in place, so that we know who is accessing the data. This data needs to be anonymized to the point that it's impossible to track back to the user or system on which it was generated. This end to end protection model needs to be analyzed that if the security controls were ever circumvented by the attacker what would be the most that they could gain from this data.

3.1.2.12 Security Test Plan

All the above security requirement will have to go into the product Security Test Plan which needs to be reviewed by the product architect and team and signed off.

To assess this level, the Security Engineer needs to do a risk assessment of the above mentioned requirements and complete the following tasks:

- Prioritize all the above mentioned requirements based on the assessment.
- Identify the "Minimum Shipment" requirement and executed them on priority.
- Fulfill the DoD criteria to mark the requirements as completed.
- Redo the CVSS scoring for all identified security issues by considering parameters like environment, impact, exposure, and distribution. This will either increase or decrease the risk.
- Address all critical risk flaws identified at CVSS scoring immediately followed by High, medium and low risk.
- Perform risk analysis for security flaws which demand 'Design Change' or 'Architectural Change'.
- All 'Design Change' or 'Architectural Change' high risk items need to be converted into User Stories and implemented on priority.
- Replace all third-party vulnerable libraries with safer versions.
- Prioritize the fix for vulnerabilities reported by vulnerability scanner.
- Address all high risk security flaws found in the static analysis.
- Uninstall the unwanted services and software from the system.
- Make sure that all entry points have authentication, authorization and identity management.
- Document the false positives of vulnerability scans, static analysis, and 3rd party library reviews.
- All security defects found at code review need to be immediately analyzed and addressed before code check-ins.
- All sensitive data should be stored securely.
- Mitigations or mitigation plan should be in place for identified security flaws.
- Define security controls that will build a strong defense in depth for identified attack vectors.
- Respond quickly, efficiently and accurately to all security flaws reported in the field.
- Outcome of reviews / finding / learning / are converted into enforced policies, to be followed by the product team members.
- Perform risk analysis of all identified security limitation and document them.
- Updated Security Test Plan with new findings.

At this stage, the Security Engineer and the product team get a fair idea about security and different types of attacks to which the product is exposed to when input validation and input data sanitization is not taken into consideration.

The vulnerability scanners and the static analysis tools will help to identify most of the input validation flaws. It is at this stage the product will be able to handle most of the input validation attacks such as, XSS, SQL Injection, XML injection, CSRF attack, Malicious File upload attack, Client-Side Validation Flaw, Buffer Overflow, or Cookie poisoning.

3.1.3 Intermediate Level

With the Security Engineer advocating secure coding practices and the development team following the security patterns during the development cycle, the product team is now ready and moves to the Intermediate level of the product security maturity model.

At this Level the Security Engineer and the product team should start working on the following security requirements.

3.1.3.1 Secure Functional Requirements

Every functional requirement created during the product development should have a security component in it to address the security concerns of the feature. The Security Engineer has to be a part of every design and planning meetings to advocate product security and to identify security design flaws.

3.1.3.2 Cryptographic Support Requirement

In this review the following areas need to be addressed

- Use strong ciphers for SSL communication and avoid weak ciphers like null & export ciphers.
- Use SHA2 for certificate generation.
- Implement certificate revocation module.
- Implement certificate verification module.
- At minimum, keys size should be >= 2048 bit.
- No hardcoded keys should be used in the encryption module
- Recycle Crypto keys periodically.
- On failure, SSL module should fail securely and should not fall back to plain text.

3.1.3.3 Least Privilege Implementation

Least privilege implementation helps to create roles and permission to implement the least and the required amount of privileges to get all functionalities are working as expected.

Data classification is implemented in this requirement so that the data collected or generated by the system is classified and put into buckets that have different levels of privileges assigned to them.

3.1.3.4 Threat Modeling

Threat Modeling is an organized procedure to optimize the security of the product by identifying security objectives and vulnerabilities, and then defining countermeasures to prevent or mitigate the effects of threats.

Threat modeling requires the creation of Data Flow Diagrams (DFD) by the Security Engineer and reviewed by the Product Architects. It is good to have threat modeling done at the Intermediate product security maturity level because at this stage the security engineer will have a clear idea of security and internals of product implementation. This will help in creation of accurate DFD’s and effective Threat Model.

3.1.3.5 Secure Architecture Review

Product architecture needs to go through regular cycles of security review. With the ever-evolving attack patterns, the existing defense can fall weak. Hence all the existing input validation controls need to be reviewed and fixed to handle the new attack vectors.

The product needs to be dissected into different layers and new security controls need to be integrated at each layers if not present. This will ensure multiple levels of defense system in the product. Hence if the attacker circumvents one line of defense, he still needs to break the next level of the defense.

The possible layers are

- **Client side** - Here mostly the security control will be in the form of JavaScript to restrict malicious inputs.
- **Server side** - Depending on the technology and design, there can be multiple layers and security controls you need to place at each of these layers. For example, the 'parameterized SQL query' takes care of SQL Injection and escaping functions which will take care of XSS and SQL Injection.
- **Database side** - Security Controls need to be placed to validate the type and size of data before storing in the database. It will also validate the privileges of the user performing the operation.

3.1.3.6 Security Test Plan

All the above security requirements are subjected to risk analysis, prioritization and ranking, and need to be updated into the Security Test Plan.

All issues that are uncovered through above mentioned security requirements need to get converted into User Stories or security backlog. This security backlog needs to be cleared to move to the mature level of product security maturity model. The Security Engineer needs to assess this level as he did in the Basic level by rating the executed security requirements.

3.1.4 Mature Level

At the mature level of the maturity model the Security Engineer and the product team has a good understanding of product security and the product is clean from all low hanging security issues. It also can handle most of the attacks such as OWASP Top 10, DoS attacks, SSL attacks etc. With in-depth understanding of product security, the security engineer now starts executing the following security requirements.

3.1.4.1 Product Fuzzing

Fuzzing is a negative testing which involves throwing all possible types of data at the identified entry points to find security flaws.

Fuzzing can be done in two ways:

- **Dump fuzzing**: The fuzzing tool does the job of throwing random data at the entry points of the product and it is done without understanding the structure of the data the product consumes.
- **Smart Fuzzing**: This type of fuzzing needs in-depth understanding of the data format consumed by the services/processes. For example, Peach Fuzzer requires expertise in PIT file creation. When using the PIT files, the user should be able to control the fuzzing at any layer of the product. The other good fuzzing tool is Codenomicon fuzzer.

3.1.4.2 Reverse-Engineering

This requirement will help to determine if attackers can reverse engineer the product to bypass licenses validation, crack password, establishing rogue entry into the system, or retrieve sensitive data by circumventing the existing security controls and counter measures.

3.1.4.3 Writing Exploit

Executing this requirement requires a lot of expertise and deep understanding of the system and years of experience in writing exploits and experience of tools like Metasploit. It will need the Security Engineer to identify a vulnerability which is exploitable, then execute the malicious payload on the victim system to remotely control it.

4. Conclusion

This product security maturity model helps organizations to proceed with the four levels of maturity management and defines a roadmap from the current state of product security to its desired state. At each level of the maturity model, the team will be able to demonstrate measurable results by rating the requirements. The defined proactive approach will help to define and achieve milestones focused on reducing the product risk and helps to identify the risks earlier in the SDL.

This maturity model will build multiple defense layers in the product and make it extremely difficult for the Bad Guys to break through. The effectiveness of this model can be seen at every security level because the team addresses security earlier in the development stage. Over the period of time, the security defect rate can reach to near-zero bug.

Appendix

Security requirement rating parameters examples

	Resources Budgeting	**Static Analysis**
1 - None	Not properly resourced	Use no Static analysis tools or use compiler flags.
2 - Initial	Security engineer headcount is approved	Static analysis done before the code commit
3 - Basic	Security Engineer is identified and he is briefed on his job responsibilities	Static analysis runs automatically with builds
4 - Acceptable	Security Engineer actively works on his job responsibilities	Product is analyzed frequently; defect rate is decreasing; reported findings are triaged
5 - Mature	Have a seasoned Product Security Architect (PSA) dedicated for each product	Fixed the defects; Enable the required security checks to reduce noise; Real defect rate is near to zero (0)
	Mandatory Security Trainings / Certification	**Fuzz Testing**
1 - None	Have little or no product security training	Fuzz testing not performed
2 - Initial	Have identified and use free product security courses	Free tools are used
3 - Basic	Mandatory security courses identified; Assigned training completed	Security Engineer / Team is trained on fuzzing tools (E.g. Peach and Codenomicon)
4 - Acceptable	Major topics are delivered and internal processes are covered	All products are fuzzed frequently; new custom scripts are created;
5 - Mature	Keep program current and have internal Subject Matter Expert (SME)	Controlled fuzzing gives ability to fuzz at different layers of the product.
	Security Tools	**Secure Coding Standards**
1 - None	Have little or no security tools are identified	No secured coding standards
2 - Initial	Have identified and used free security tools	Aware of standards but occasional adherence
3 - Basic	Procured the required licensed security tools and apply them appropriately.	Adopted appropriate standards
4 - Acceptable	Tools are integrated into the build system and to run on regular basis	Following adopted standards
5 - Mature	Custom security tools developed to close the identified gaps	Standards are integrated into manual code reviews and static analysis
	Product Hardening	**Open Source / Third Party Libraries**
1 – None	Product running with the default configuration, unwanted software's and services.	Selected by any engineer; used with no constraints
2 – Initial	Product is reviewed for open ports, default configuration, unwanted software's etc.	Manually maintain lists of used binaries
3 – Basic	Product hardening exercise is completed and the security issues are addressed	Run inventory tools (BlackDuck) and tools like dependency checker
4 - Acceptable	Every new component that is added to the system is reviewed.	Fully maintaining all documented Third party libraries; Replace vulnerable Libraries with safer versions.

5 – Mature	Any change to the system can open an entry point and it must be reviewed and approved by the Security Engineer and Product Security Architect (PSA)	All Open Source / Third Party Libraries integrated and used securely.
	Vulnerability Scans / Penetration Testing	**Code Review**
1 – None	Escalations from customers	Ad-hoc; Without security patterns in mind
2 – Initial	Vulnerability scanners / tools are evaluated, identified, and used by Security Engineer	Aware of standards, occasional adherence
3 – Basic	Vulnerability scans are performed regularly and defects are analyzed	Conducted on risky new code by multiple engineers
4 - Acceptable	Pentesting expert available; defects are addressed; findings are used to improve early defect finding	Conducted on all potentially risky code using a shared tool
5 – Mature	Every release is pentested by an external pentesters; defects are fixed before the product launch	Conducted regularly using a code sharing collaboration tool (e.g. SmartBear Collaborator)

References

Examples of the big data breaches:

- http://www.forbes.com/sites/moneybuilder/2015/01/13/the-big-data-breaches-of-2014/
- http://www.networkworld.com/article/2864382/microsoft-subnet/hackers-released-xbox-one-sdk-claimed-unreleased-games-may-be-leaked.html
- https://www.opm.gov/cybersecurity/

Security Literature:

- https://tysonmax20042003.wordpress.com/tag/types-of-exploits/

Definition of Done:

- http://scrumguides.org/scrum-guide.html#artifact-transparency-done

Defense in depth:

- https://en.wikipedia.org/wiki/Defense_in_depth_(computing)

Common Vulnerability Scoring System:

- https://www.first.org/cvss/calculator/3.0
- https://nvd.nist.gov/cvss.cfm?calculator&version=2

Examples of Security Requirements:

- https://security.web.cern.ch/security/recommendations/en/checklist_for_coders.shtml

Training / Certification / Tools:

- https://www.owasp.org
- http://projects.webappsec.org/f/WASC-TC-v2_0.pdf
- http://www.eccouncil.org/Certification/certified-ethical-hacker
- https://www.offensive-security.com/metasploit-unleashed-training/metasploit-unleashed-mastering-the-framework/

Examples of Threat Agents and Methods:

- https://ics-cert.us-cert.gov/content/cyber-threat-source-descriptions
- http://www.lovemytool.com/files/vulnerabilities-threats-and-attacks-chapter-one-7.pdf